React Native has two ways to initiate a project:

React Native CLI

Expo CLI

For React Native CLI

We need to set Up Android and/or iOS Development Environment

For Android (Requires Android Studio):

Set up the Android environment variables

Create an Android Virtual Device (AVD):

Similar is the case for IOS

React Native CLI Project in GitHub Codespaces

The biggest challenge of running React Native in GitHub Codespaces is testing on a device or emulator.

GitHub Codespaces doesn't support running Android emulators directly since it doesn't have access to the local graphical environment or Android Studio. We won't be able to launch the Android emulator directly from within Codespaces.

GitHub Codespaces is an excellent tool for writing code, managing projects, and collaborating on React Native apps, especially for those who want to work in a cloud-based development environment. However, due to the complexity of emulators and native dependencies (like Android Studio or Xcode), running the app on an emulator isn't feasible in Codespaces.

Unfortunately, we cannot directly run Android Emulators or Android Studio within GitHub Codespaces because Codespaces does not support graphical environments or hardware acceleration needed to run emulators. Android Emulators require hardware-level virtualization support and a graphical interface, which isn't available in the containerized and cloud-based environment that GitHub Codespaces operates in.

Approaches:

Remote Development

Write and Code in GitHub Codespaces:

Clone the repository locally to your computer, run the emulator (Android/iOS), and test the app there.

However, this approach is not feasible as we don't wont other to clone the code.

Approach 2:

If we want to test and develop directly in the cloud, Expo might be a better choice for cloud-based testing, since it allows you to test your app directly on a physical device using the Expo Go app without needing an emulator.

Option to convert React Native CLI to Expo Cli

Converting a React Native CLI project to Expo is possible, but it involves replacing non-Expo dependencies with Expo equivalents and refactoring code where necessary. The main challenge comes from native dependencies that Expo doesn't support, but Expo does provide many built-in APIs that cover a wide range of use cases.

For instance:

- Replace react-native-camera with Expo's expo-camera.
- Replace react-native-maps with expo-maps.

However, if you find that your project requires custom native code that Expo does not support, you can always eject from Expo later. Ejecting gives you full control over the native project but removes the managed Expo environment.